0.1 Support Vector Machines (SVM)

0.1.1 Linear classifiers with large margins

In this section we briefly outline the basic idea behind Support Vector Machines (SVM). SVMs have been very popular a few years ago, and while we have no systems that can outperform them, they are still a great workhorse for many applications. I certainly recommend them as one of the first learning machines I would try on many applications. SVMs, and the underlying statistical learning theory, has been worked out by Vladimir Vapnik since the early 1960, but some breakthroughs were also made in the late 1990 with some collaborators like Corinna Cortes, Chris Burges, Alex Smola, and Bernhard Schölkopf to name but a few.

While I outline some of the underlying formulas, I will not follow all the steps in detail here. The main purpose here is to state the main formulas and to give an idea of the underlying concepts. There are several good tutorials on the web with more details. Here I mainly want to provide the big picture, though I need to show some formulas to highlight some of the discussion.

The basic SVMs are concerned with binary classification. Figure 0.1 shows an example of two classes, depicted by different symbols, in a two dimensional attribute space. We distinguish here attributes from features as follows. Attributes are the raw measurements, where as features can be made up by combining attributes. For example, the attributes x_1 and x_2 could be combines in a feature vector $(x_1, x_1x_2, x_2, x_1^2, x_2^2)^T$. This will become important later, but it is important to introduce the notation here. Our training set consists again of m data with attribute values $\mathbf{x}^{(i)}$ and labels $y^{(i)}$. We chose here the labels of the two classes as $y \in \{-1, 1\}$, as this will nicely simplify some equations.

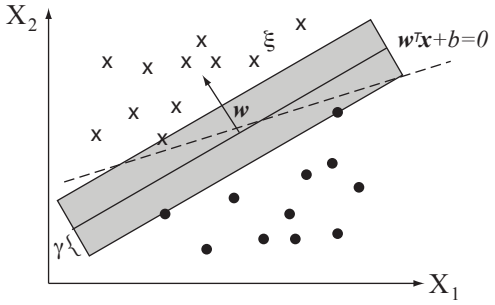

Fig. 0.1 Illustration of linear support vector classification.

The two classes in the figure 0.1 can be separated by a line, which can be parameterized by

$$w_1x_1 + w_2x_2 + b = \mathbf{w}^T\mathbf{x} + b = 0. \tag{0.1}$$

While the first equation shows the lines equation with its components in two dimensions, the next expression is the same in any dimension. Of course, in three dimension we would talk about a plane. In general, we will talk about a **hyperplane** in any dimensions. The particular hyperplane is the dividing or separating hyperplane between

the two classes. I also show the **margin** γ in the figure, which is the perpendicular distance between the dividing hyperplane and the closest point.

The main point to realize now is that the dividing hyperplane that maximizes the the margin, the so called **maximum margin classifier**, is the best solution we can find. Why is that? We should assume that the training data, shown in the figure, are some unbiased examples of the true underlying density function describing the distribution of points within each class and thus representative of the most likely data. It is then likely that new data points, which we want to classify, are close to the already existing data points. Hence, if we make the separating hyperplane as far as possible from each point, than it is most likely to not make wrong classification. Or, with other words, a separating hyperplane like the one shown as dashed line in the figure, is likely to generalize much worse than the maximum margin hyperplane. So the maximum margin hyperplane is the best generalizer for binary classification for the training data.

What is if we can not divide the data with a hyperplane and we have to consider non-linear separators. Don't we then run into the same problems as outlined before, specifically the bias-variance tradeoff? Yes, indeed, this will still be the challenge, and our aim is really to work on this problem. But before going there it is useful to formalize the linear separable case in some detail as the representation of the optimization problem will be a key in applying some tricks later.

Learning a linear maximum margin classifier on labeled data means finding the parameters (w) and b that maximizes the margin. For this we need to compute the distances of the closest point to the hyperplane, which is

$$\tilde{\gamma}^{(i)} = y^{(i)} \left(\mathbf{w}^T \mathbf{x}^{(i)} + b \right). \tag{0.2}$$

At this point it is useful to normalize this equation as

$$\gamma^{(i)} = y^{(i)} \left((\frac{\mathbf{w}}{||\mathbf{w}||})^T \mathbf{x}^{(i)} + \frac{b}{||\mathbf{w}||} \right). \tag{0.3}$$

The vector $\mathbf{w}/||\mathbf{w}||$ is the normal vector of the hyperplane, a vector of unit length perpendicular to the hyperplane ($||\mathbf{w}||$ is the Euclidean length of the vector $\mathbf{w}$. The overall margin we want to maximize is the distance to the closest point,

$$\gamma = \min_i \gamma^{(i)}. \tag{0.4}$$

By looking at equation 0.3 we see that maximizing γ is equivalent to minimizing $||\mathbf{w}||$, or, equivalently, of minimizing

$$\min_{\mathbf{w},b} \frac{1}{2} ||\mathbf{w}||^2. \tag{0.5}$$

More precisely, we want to maximize this margin under the constraint that no training data lies within the margin,

$$\mathbf{w}^T \mathbf{x}^{(i)} + b \geq 1 \quad \text{for} \quad y^{(i)} = 1 \tag{0.6}$$

$$\mathbf{w}^T \mathbf{x}^{(i)} + b \leq -1 \quad \text{for} \quad y^{(i)} = -1, \tag{0.7}$$

which can nicely be combines with our choice of class representation as

$$-y^{(i)}(\mathbf{w}^T\mathbf{x}+b)-1\leq 0. \tag{0.8}$$

Thus we have a quadratic minimization problem with linear inequalities as constraint. Taking a constrain into account can be done with a **Lagrange formalism**. For this we simply add the constraints to the main objective function with parameters α_i called Lagrange multipliers,

$$Ł^P(\mathbf{w},b,\alpha_i)=\frac{1}{2}||\mathbf{w}||^2-\sum_{i=1}^{m}\alpha_i[y^{(i)}(\mathbf{w}^T\mathbf{x}+b)-1]. \tag{0.9}$$

The Lagrange multipliers determine how well the constrain are observed. In the case of $\alpha_i=0$, the constrains do not matter. In order conserve the constrains, we should thus make these values as big as we can. Finding the maximum margin classifier is given by

$$p^*=\min_{\mathbf{w},b}\max_{\alpha_i}Ł^P(\mathbf{w},b,\alpha_i)\leq p^*=\max_{\alpha_i}\min_{\mathbf{w},b}Ł^D(\mathbf{w},b,\alpha_i)=d^*. \tag{0.10}$$

In this formula we also added the formula when interchanging the min and max operations, and the reason for this is the following. It is straight forward to solve the optimization problem on the left hand side, but we can also solve the related problem on the right hand side which turns out to be essential when generalizing the method to nonlinear cases below. Moreover, the equality hold when the optimization function and the constraints are convex[1]. So, if we minimize Ł by looking for solutions of the derivatives $\frac{\partial Ł}{\partial \mathbf{w}}$ and $\frac{\partial Ł}{\partial b}$, we get

$$\mathbf{w}=\sum_{i=1}^{m}\alpha_i y^{(i)}\mathbf{x}^{(i)} \tag{0.11}$$

$$0=\sum_{i=1}^{m}\alpha_i y^{(i)} \tag{0.12}$$

Substituting this into the optimization problem we get

$$\max_{\alpha_i}\sum_i\alpha_i-\frac{1}{2}\sum_{i,j}y^{(i)}y^{(y)}\alpha_i\alpha_j\mathbf{x}^{(i)T}\mathbf{x}^{(j)}, \tag{0.13}$$

subject to the constrains

$$\alpha_i\geq 0 \tag{0.14}$$

$$\sum_{i=1}^{m}\alpha_i y^{(i)}=0. \tag{0.15}$$

From this optimization problem it turns out that the α_i's of only a few examples, those ones that are lying on the margin, are the only ones with have $\alpha_i\neq 0$. The corresponding training examples are called **support vectors**. The actual optimization

[1] Under these assumptions there are other conditions that hold, called the **Karush-Kuhn-Tucker** conditions, that are useful in providing proof in the convergence of these the methods outlined here.

can be done with several algorithms. In particular, John Platt developed the sequential minimal optimization (SMO) algorithm that is very efficient for this optimization problem. Please note that the optimization problem is convex and can thus be solved very efficiently without the danger of getting stuck in local minima.

Once we found the support vectors with corresponding α_i's, we can calculate (w) from equation 0.11 and b from a similar equation. Then, if we are given a new input vector to be classified, this can then be calculated with the hyperplane equation 0.1 as

$$y = \begin{cases} 1 & \text{if} \quad \sum_{i=1}^{m} \alpha_i y^{(i)} \mathbf{x}^{(i)T} \mathbf{x} > 0 \\ -1 & \text{otherwise} \end{cases} \tag{0.16}$$

Since this is only a sum over the support vectors, which should be only a few data points from the training set, classification becomes very efficient after training.

0.1.2 Soft margin classifier

So far we only discussed the linear separable case. But how about the case when there are overlapping classes? It is possible to extend the optimization problem by allowing some data points to be in the margin while penalizing these points somewhat. We include therefore some **slag variables** ξ_i that reduce the effective margin for each data point, but we add to the optimization a penalty term that penalizes if the sum of these slag variables are large,

$$\min_{\mathbf{w},b} \frac{1}{2}||\mathbf{w}||^2 + C \sum_i \xi_i, \tag{0.17}$$

subject to the constrains

$$y^{(i)}(\mathbf{w}^T \mathbf{x} + b) \geq 1 - \xi_i \tag{0.18}$$

$$\xi_i \geq 0 \tag{0.19}$$

The constant C is a free parameter in this algorithm. Making this constant large means allowing less points to be in the margin. This parameter must be tuned and it is advisable to at least try to vary this parameter to verify that the results do not dramatically depend on a initial choice.

0.1.3 Nonlinear Support Vector Machines

We have treated the case of overlapping classes while assuming that the best we can do is still a linear separation. But what if the underlying problem is separable, f only with a more complex function. We will now look into the non-linear generalization of the SVM.

When discussing regression we started with the linear case and then discussed non-linear extensions such as regressing with polynomial functions. For example, a linear function in two dimensions (two attribute values) is given by

$$y = w_0 + w_1 x_1 + w_2 x_2, \tag{0.20}$$

and an example of a non-linear function, that of an polynomial of 3rd order, is given by

$$y = w_0 + w_1 x_1 + w_2 x_2 + w_3 x_1 x_2 + w_4 x_1^2 + w_5 x_2^2. \tag{0.21}$$

The first case is a linear regression of a feature vector

$$\mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}. \tag{0.22}$$

We can also view the second equation as that of linear regression on a feature vector

$$\mathbf{x} \to \phi(\mathbf{x}) = \begin{pmatrix} x_1 \\ x_2 \\ x_1 x_2 \\ x_1^2 \\ x_2^2 \end{pmatrix}, \tag{0.23}$$

which can be seen as a mapping $\phi(\mathbf{x})$ of the original attribute vector. We call this mapping a **feature map**. Thus, we can use the above maximum margin classification method in non-linear cases if we replace all occurrences of the attribute vector $\mathbf{x}$ with the mapped feature vector $\phi(\mathbf{x})$. There are only two problems remaining. One is that we have again the problem of overfitting as we might use too many feature dimensions and corresponding free parameters w_i. The second is also that with an increased number of dimensions, the evaluation of the equations becomes more computational intensive. However, there is a great trick to alleviate the later problem in the case when the methods only rely on dot products, like in the case of the formulation in the dual problem. In this, the function to be minimized, equation 0.13 with the feature maps, only depends on the dot products $\phi(\mathbf{x}^{(i)})^T \phi(\mathbf{x}^{(j)})$. Also, when predicting the class for a new input vector $\mathbf{x}$ from equation 0.11 when adding the feature maps, we only need the resulting values for the dot products $\phi(\mathbf{x}^{(i)})^T \phi(\mathbf{x})$ which can sometimes be represented as function called **Kernel function**,

$$K(\mathbf{x}, \mathbf{z}) = \phi(\mathbf{x})^T \phi(\mathbf{z}). \tag{0.24}$$

Instead of actually specifying a feature map, which is often a guess to start, we could actually specify a Kernel function. For example, let us consider a quadratic feature map

$$K(\mathbf{x}, \mathbf{z}) = (\mathbf{x}^T \mathbf{z} + c)^2. \tag{0.25}$$

We can then try to write this in the form of equation 0.24 to find the corresponding feature map. That is

$$\begin{aligned} K(\mathbf{x}, \mathbf{z}) &= (\mathbf{x}^T \mathbf{z})^2 + 2c\mathbf{x}^T \mathbf{z} + c^2 && (0.26) \\ &= (\sum_i x_i z_i)^2 + 2c \sum_i x_i z_i + c^2 && (0.27) \\ &= \sum_j \sum_i (x_i x_j)(z_i z_j) + \sum_i (\sqrt{(}2c) x_i)(\sqrt{(}2c) z_i) + cc && (0.28) \\ &= \phi(\mathbf{x})^T \phi(\mathbf{z}), && (0.29) \end{aligned}$$

with

$$\phi(\mathbf{x}) = \begin{pmatrix} x_1 x_1 \\ x_1 x_2 \\ \dots \\ x_n x_1 \\ x_n x_2 \\ \dots \\ \sqrt{2c} x_1 \\ \sqrt{2c} x_2 \\ \dots \\ c \end{pmatrix}, \tag{0.30}$$

The dimension of this feature vector is $O(n^2)$ for n original attributes. Thus, evaluating the dot product in the mapped feature space is much more time consuming then calculating the Kernel function which is just the square of the dot product of the original attribute vector. The dimensionality Kernels with higher polynomials is quickly rising, making the benefit of the Kernel method even more impressive.

While we have derived the corresponding feature map for a specific Kernel function, this task is not always easy and not all functions are valid Kernel functions. We have also to be careful that the Kernel functions still lead to convex optimization problems. In practice, only a small number of Kernel functions is used. Besides the polynomial Kernel mention before, one of the most popular is the Gaussian Kernel,

$$K(\mathbf{x}, \mathbf{z}) = \exp -\frac{||\mathbf{x} - \mathbf{z}||^2}{2\gamma^2}, \tag{0.31}$$

which corresponds to an infinitely large feature map.

As mentioned above, a large feature space corresponds to a complex model that is likely to be prone to overfitting. We must therefore finally look into this problem. The key insight here is that we are already minimizing the sum of the components of the parameters, or more precisely the square of the norm $||\mathbf{w}||^2$. This term can be viewed as **regularization** which favours a smooth decision hyperplane. Moreover, we have discussed two extremes in classifying complicated data, one was to use Kernel functions to create high-dimensional non-linear mappings and hence have a high-dimensional separating hyperplane, the other method was to consider a low-dimensional separating hyperplane and interpret the data as overlapping. The last method includes a parameter C that can be used to tune the number of data points that we allow to be within the margin. Thus, we can combine these two approaches to classify non-linear data with overlaps where the soft margins will in addition allow us to favour more smooth dividing hyperplanes.

0.1.4 Regularization and parameter tuning

In practice we have to consider several free parameters when applying support vector machines. First, we have to decide which Kernel function to use. Most packages have a number of choices implemented. We will use for the following discussion the Gaussian Kernel function with width parameter γ. Setting a small value for γ and allowing for a large number of support vectors (small C), corresponds to a complex model. In contrast, larger width values and regularization constant C will increase the stiffness of the model and lower the complexity. In practice we have to tune these parameters to

get good results. To do this we need to use some form of validation set, as discussed in section **??**, and k-times cross validation is often implemented in the software packages. An example of the SVM performance (accuracy) on some examples (Iris Data set from the UCI repository; From Broadman and Trappenberg, 2006) is shown in figure 0.2 for several values of γ and C. It is often typical that there is a large area where the SVM works well and has only little variations in terms of performance. This robustness has helped to make SVMs practical methods that often outperform other methods. However, there is often also an abrupt onset of the region where the SVM fails, and some parameter tuning is hence required. While just trying a few settings might be sufficient, some more systematic methods such as grid search or simulated annealing also work well.

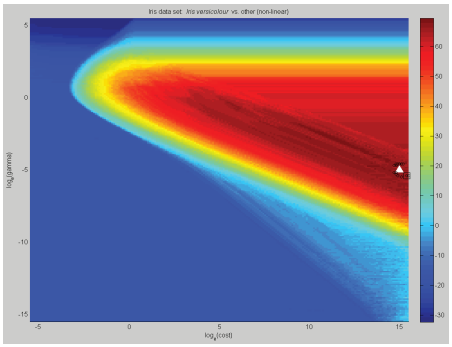

Fig. 0.2 Illustration of SVM accuracy for different values of paraneters C abd γ.

0.1.5 Statistical learning theory and VC dimension

SVMs are good and practical classification algorithms for several reasons, including the advantage of being convex optimization problem that than can be solved with quadratic programming, have the advantage of being able to utilize the Kernel trick, have a compact representation of the decision hyperplane with support vectors, and turn out to be fairly robust with respect to the hyper parameters. However, in order to be good learners, they need to moderate the variance-bias tradeoff dicussed in section **??**. A great theoretical contributions of Vapnik and colleagues was the embedding of supervised learning into statistical learning theory and to derive some bounds that make statements on the average ability to learn form data. We outline here briefly the ideas and state some of the results. We discuss this issue here in the context of binary classification, although similar observations can be made in the case of multiclass classification and regression.

We start again by stating our objective, which is to find a hypothesis which minimized the generalization error. To state this a bit more differentiated and to use the nomenclature common in these discussions, we call the error function here the **risk function** R. In particular, the **expected risk** for a binary classification problem is the probability of misclassification,

$$R(h) = P(h(x) \neq y) \tag{0.32}$$

Of course, we generally do not know this density function, though we need to approximate this with our validation data. We assume thereby again that the samples are iid (independent and identical distributed) data, and can then estimate what is called the **empirical risk**,

$$\hat{R}(h) = \frac{1}{m} \sum_{i=1}^{m} \mathbb{1}(h(\mathbf{x}^{(i)}; \theta) = y^{(i)}). \tag{0.33}$$

We use here again m as the number of examples, but note that this is here the number of examples in the validations set, which is the number of all training data minus the ones used for training. Also, we will discuss this empirical risk further, but note that it is better to use the regularized version that incorporates a smoothness constrain such as

$$\hat{R}_{rmreg}(h) = \frac{1}{m} \sum_{i} \mathbb{1}(h(\mathbf{x}^{(i)}; \theta) = y^{(i)}) - \lambda ||\mathbf{w}||^2 \tag{0.34}$$

in the case of SVM, where λ is a regularization constant. Thus, wherever $\hat{R}(h)$ is used in the following, we can replace this with $\hat{R}_{rmreg}(h)$. **Empirical risk minimization** is the process of finding the hypothesis $\hat{h}$ that minimizes the empirical risk,

$$\hat{h} = arg \min_{h} \hat{R}(h). \tag{0.35}$$

The empirical risk is the MLE of the mean of a Bernoulli-distributed random variable with true mean $R(h)$. Thus, the empirical risk is itself a random variable for each possible hypothesis h. Let us first assume that we have k possible hypothesis h_i. We now draw on a theorem by Hoeffding called the **Hoeffding inequality** that provides and upper bound for the sum of random numbers to its mean,

$$P(|R(h_i) - \hat{R}(h_i)| > \gamma) \leq 2 \exp(-2\gamma^2 m). \tag{0.36}$$

This formula states that there is a certain probability that we make an error larger than γ for each hypothesis of the empirical risk compared to the expected risk, although the good news is that this probability is bounded and that the bound itself becomes exponentially smaller with the number of validation examples. This is already an interesting results, but we now want to know the probability that some, out of all possible hypothesis, are less than γ. Using the fact that the probability of the union of several events is always less or equal to the sum of the probabilities, one can show that with probability $1 - \delta$ the error of a hypothesis is bounded by

$$|R(h) - \hat{R}(h)| \leq \sqrt{\frac{1}{2m} \log \frac{2k}{\delta}}. \tag{0.37}$$

This is a great results since it shows how the error of using an estimate the risk, the empirical risk that we can evaluate from the validation data, is getting smaller with training examples and with the number of possible hypothesis.

While the error scales only with the log of the number of possible hypothesis, the values goes still to infinite when the number of possible hypothesis goes to infinite, which much more resembles the situation when we have parameterized hypothesis. However, Vapnik was able to show the following generalization in the infinite case,

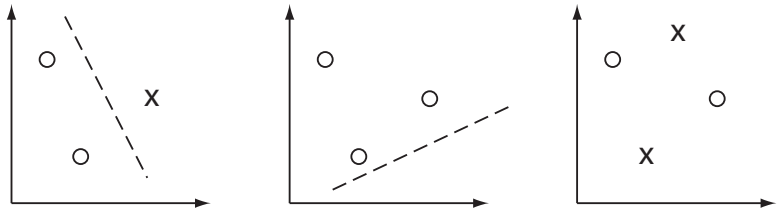

Fig. 0.3 Illustration of VC dimensions for the class of linear functions in two dimensions.

which is that given a hypothesis space with **Vapnic-Chervonencis** dimension VC($\{\mathrm{h}\}$), then, with probability $1-\delta$, the error of the empirical risk compared to the expected risk (true generalization error) is

$$|R(h) - \hat{R}(h)| \leq O\left(\sqrt{\frac{VC}{m}\log\frac{m}{VC} + \frac{1}{m}\log\frac{1}{\delta}}\right). \tag{0.38}$$

The VC dimensions is thereby a measure of how many points can be divided by a member of the hypothesis set for all possible label combinations of the point. For example, consider three arbitrary points in two dimensions as shown in figure 0.3, and let us consider the hypothesis class of all possible lines in two dimensions. I can always divide the three points under any class membership condition, of which two examples are also shown in the figure. In contrast, it is possible to easily find examples with four points that can not be divided by a line in two dimensions. The VC dimension of lines in two dimensions is hence $VC = 3$.[2]

0.2 SV-Regression and implementation

0.2.1 Support Vector Regression

While we have mainly discussed classification in the last few sections, it is time to consider the more general case of regression and to connect these methods to the general principle of maximum likelihood estimation outlined in the previous chapter. It is again easy to illustrate the method for the linear case before generalizing it to the non-linear case similar to the strategy followed for SVMs.

We have already mentioned in section **??** the ϵ-insensitive error function which does not count deviations of data from the hypothesis that are less than ϵ form the hypothesis, This is illustrated in figure 0.4. The corresponding optimization problem is

$$\min_{\mathbf{w},b} \frac{1}{2}||\mathbf{w}||^2 + C\sum_i(\xi_i + \xi^*), \tag{0.39}$$

subject to the constrains

$$y^{(i)} - \mathbf{w}^T\mathbf{x} - b \leq \xi_i \tag{0.40}$$

[2] Three points of different classes can not be separated by a single line, but these are singular points that are not effective in the definition of VC dimension.

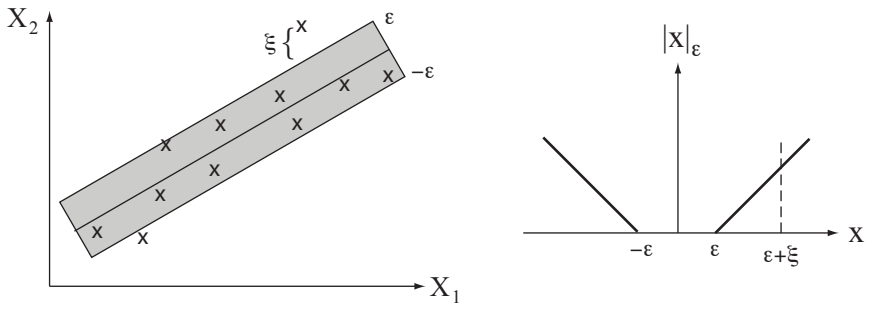

Fig. 0.4 Illustration of support vector regression and the ϵ-insensitive cost function.

$$y^{(i)} - \mathbf{w}^T\mathbf{x} - b \geq \xi_i^* \tag{0.41}$$

$$\xi_i, \xi_i^* \geq 0 \tag{0.42}$$

The dual formulations does again only depend on scalar products between the training examples, and the regression line can be also be expressed by a scalar product between the support vectors and the prediction vector,

$$h(\mathbf{x}; \alpha_i, \alpha_i^*) = \sum_{i=1}^{m} (\alpha_i - \alpha_i^*)\mathbf{x}_i^T\mathbf{x}. \tag{0.43}$$

This, we can again use Kernels to generalize the method to non-linear cases.

0.2.2 Implementation

There are several SVM implementations available, and SVMs are finally becoming a standard component of data mining tools. We will use the implementation called LIBSVM which was written by Chih-Chung Chang and Chih-Jen Lin and has interfaces to many computer languages including Matlab. There are basically two functions that you need to use, namely `model=svmtrain(y,x,options)` and `svmpredict(y,x,model,options)`. The vectors `x` and `y` are the training data or the data to be tested. `svmtrain` uses k-fold cross validation to train and evaluate the SVM and returns the trained machine in the structure `model`. The function `svmpredict` uses this model to evaluate the new data points. Below is a list of options that shows the implemented SVM variants. We have mainly discussed C-SVC for the basic soft support vector classification, and epsilonSVR for support vector regression.

```
-s svm_type : set type of SVM (default 0)
0 -- C-SVC
1 -- nu-SVC
2 -- one-class SVM
3 -- epsilon-SVR
4 -- nu-SVR
-t kernel_type : set type of kernel function (default 2)
0 -- linear: u'*v
1 -- polynomial: (gamma*u'*v + coef0)^degree
```

```
2 -- radial basis function: exp(-gamma*|u-v|^2)
3 -- sigmoid: tanh(gamma*u'*v + coef0)
-d degree : set degree in kernel function (default 3)
-g gamma : set gamma in kernel function (default 1/num_features)
-r coef0 : set coef0 in kernel function (default 0)
-c cost : set the parameter C of C-SVC, epsilon-SVR, and nu-SVR (default 1)
-n nu : set the parameter nu of nu-SVC, one-class SVM, and nu-SVR (default 0.5)
-p epsilon : set the epsilon in loss function of epsilon-SVR (default 0.1)
-m cachesize : set cache memory size in MB (default 100)
-e epsilon : set tolerance of termination criterion (default 0.001)
-h shrinking: whether to use the shrinking heuristics, 0 or 1 (default 1)
-b probability_estimates: whether to train a SVC or SVR model for probability estimates, 0 o
-wi weight: set the parameter C of class i to weight*C, for C-SVC (default 1)

The k in the -g option means the number of attributes in the input data.
```